Aula Exercícios Polimorfismo

```
import java.text.DecimalFormat;

public abstract class Conta {
	protected int numero;
	protected Cliente dono;
	protected double saldo;

	Conta (int numero, Cliente dono) {
		this.numero = numero;
		this.dono = dono;
		this.saldo = 0;
	}

	public void sacar(double valor) {
		saldo = saldo - valor;
	}

	public void depositar(double valor) {
		saldo = saldo + valor;
	}

	public abstract void transferir(double valor, Conta destino);


	public void setSaldo(double valor) {
		saldo = valor;
	}

	public double getSaldo() {
		return saldo;
	}

	public String toString() {
		DecimalFormat df = new DecimalFormat("#0.00");
		return "Número : " + numero + " - Cliente : " + dono.getNome() + "-" + dono.getTelefone() + " - saldo
: " + df.format(saldo);
	}

}
```

1Dada a classe abstrata **Conta**, faça:

2) criar uma classe **Cliente** com os atributos ***privates*** nome e telefone, e gerar os métodos necessários para acesso aos atributos de **Cliente**;;

3) Criar um construtor para a classe **Cliente**;

4) Criar duas subclasses de Conta: **ContaCorrente** e **ContaPoupanca**

- A classe **ContaCorrente** deverá ter os métodos **sacar** e **depositar** sobrescritos pois todo saque ou depósito de uma conta corrente deve cobrar uma taxa de R$ 0,05.
- A classe **ContaPoupanca** deve ter um método chamado **atualizaSaldo** que recebe um percentual e atualiza o saldo da conta.

5) Criar a classe **TestaConta** que declare duas referências a objetos do tipo **Conta**, sendo que uma será uma instância de **ContaCorrente** e a outra de **ContaPoupanca**.

6) Para cada conta, invoque os métodos **depositar**, **sacar** e veja os resultados. Observe as diferenças nos métodos chamados.

7) Para o objeto da **ContaPoupanca**, invoque o método **atualizaSaldo** passando um valor e veja o resultado. Observe que haverá a necessidade de usar o casting.

8) Altere a declaração das contas para seus tipos próprios, retire o casting e veja os resultados.

9) Implemente o método **transferir**

10) Crie outras contas dos dois tipos e use o método transferir entre elas. Execute e veja os resultados. Observe que o método transferir também precisa aplicar as taxas de depósito e saque nas devidas contas.

11) Crie outra subclasse de ContaCorrente que se refere à **ContaSalario**. Este tipo de conta tem uma diferenciação com relação à taxa cobrada em cada transação (saque e depósito). A taxa deixa de ser R$ 0,05 e passa a ser R$ 0,01.

12) Na classe **TestaConta**, declare contas do tipo **ContaSalario**, configure seus dados e invoque os métodos **sacar**, **depositar** e **transferir** avaliando os resultados.

13) Faça transferência entre contas de tipos diferentes e avalie os resultados. Veja se os cálculos das taxas estão corretos.

14) Mostre sempre os saldos no final de cada operação para verificar o resultado (use o toString).

Continuação – Atividade da Semana.

15) Implemente nova Classe com o método main que**:**

- Tenha uma interface para pedir os dados da Conta e do Cliente (use JoptionPane);
- Ler os dados até que o usuário digite o número da conta = 0;
- Solicitar um tipo de conta (1-Corrente, 2-Conta Salário ou 3-poupança);
- Receber os dados, instanciar um Cliente com os dados de entrada do cliente, e instanciar a conta de acordo com o tipo informado (ContaCorrente, ContaPoupanca ou ContaSalário);
- Armazenar os dados recebidos em um Array de Contas;
- Ao final do cadastro, o programa deve percorrer o Array e mostrar todas as contas cadastradas com todos os seus dados (usar o método toString);